Knowledge Discovery in Database

Report on

Stanford CoreNLP - Natural Language Processing software

horizontal line

# 

|  |  |  |
| --- | --- | --- |
| **Team Members** | **Email** | **Student Id** |
| **Akash Singh** | [asingh40@uncc.edu](mailto:asingh40@uncc.edu) | 801022198 |
| **Ankit Luthra** | [aluthra@uncc.edu](mailto:aluthra@uncc.edu) | 801019495 |
| **Bhargavi Narayan Bhat** | [bnarayan@uncc.edu](mailto:bnarayan@uncc.edu) | 801022518 |
| **Sanket Shete** | <sshete@uncc.edu> | 801018542 |

[**Introduction**](#_arolcxe0i15c)3

[**Overview**](#_kwop71b4nsuj)3

[Backbone components](#_orpzospclbcy) 4

[Architecture](#_8olzz5njpfp8) 5

[Some Useful Annotators](#_1uraz5pja6ac) 5

[**Implementation**](#_3hqpzs9cj35p)6

[Tokenizer](#_5pnwhgd4yrpf) 6

[Sentence Split](#_wrrsej1ao7fr) 7

[Part Of Speech (POS) Tagging](#_qalro2qbr0di) 7

[Lemmatization](#_y8iiz5m6r9w9) 8

[Named entity recognizer (NER)](#_iqmvw0grznsy) 8

[Coreference resolution system (coref)](#_a8sds7vt5cqw) 8

[Sentiment analysis](#_kyixx7916eym) 9

[Open Information Extraction](#_eumz84hw7yd9) 10

[**Applications**](#_5phlrvmichsf)11

[**Advantages**](#_wuub7xqiwxi0)11

[**Limitations & Mitigation**](#_2ggbzgt3pj1o)11

[**Customized Annotators**](#_sndbg3odax87)12

[**Running the java source Code**](#_9ueyk9h0cgrg)12

[**Download**](#_m246g9vvra5y)13

[Programming languages and operating systems](#_je33cksnx1m) 14

[License](#_o22awqtbgk92) 14

# 

# Introduction

Stanford CoreNLP provides a set of human language technology tools. It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, mark up the structure of sentences in terms of phrases and syntactic dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, extract particular or open-class relations between entity mentions, get the quotes people said, etc.

What is unique about Stanford CoreNLP?

* An integrated NLP toolkit with a broad range of grammatical analysis tools
* A fast, robust annotator for arbitrary texts, widely used in production
* A modern, regularly updated package, with the overall highest quality text analytics
* Support for a number of major (human) languages
* Available APIs for most major modern programming languages
* Ability to run as a simple web service

Stanford CoreNLP goal is to make it very easy to apply a bunch of linguistic analysis tools to a piece of text. A tool pipeline can be run on a piece of plain text with just two lines of code. CoreNLP is designed to be highly flexible and extensible. With a single option you can change which tools should be enabled and disabled. Stanford CoreNLP integrates many of Stanford’s NLP tools, including [the part-of-speech (POS) tagger](http://nlp.stanford.edu/software/tagger.html), [the named entity recognizer (NER)](http://nlp.stanford.edu/software/CRF-NER.html), [the parser](http://nlp.stanford.edu/software/lex-parser.html), [the Coreference resolution system](http://nlp.stanford.edu/software/dcoref.html), [sentiment analysis](http://nlp.stanford.edu/sentiment/), [bootstrapped pattern learning](http://nlp.stanford.edu/software/patternslearning.html), and the [open information extraction](http://nlp.stanford.edu/software/openie.html) tools.

# 

# Overview

## Backbone components

* **Annotation** - Annotations are the data structure which holds the results of annotators. Annotations are basically maps, from keys to bits of the annotation, such as the parse, the part-of-speech tags, or named entity tags.
* **Annotator** - Annotators are a lot like functions, except that they operate over Annotations instead of Objects. They do things like tokenize, parse, or NER tag sentences.

Annotators and Annotations are integrated by Annotation Pipelines, which create sequences of generic Annotators. To construct a Stanford CoreNLP object from a given set of properties, use Stanford CoreNLP (Properties props). This method creates the pipeline using the annotators given in the “annotators” property

## Architecture

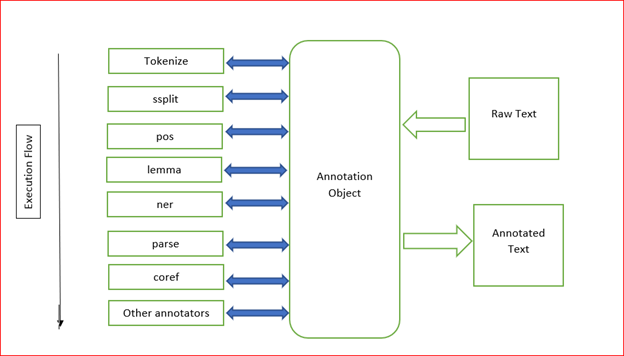


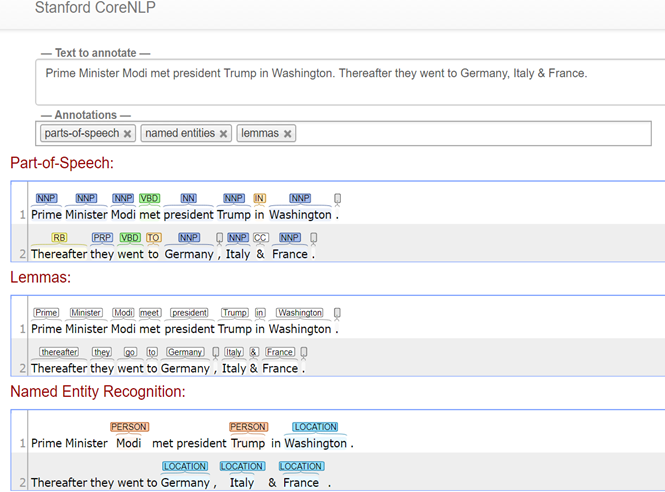
Figure 1:Overall System Architecture

Raw text is put into an Annotation object and then a sequence of Annotators add information in an analysis pipeline. The resulting Annotation, containing all the analysis information added by the Annotators, can be output in XML or plain text forms.

## Some Useful Annotators

Stanford CoreNLP integrates many of Stanford’s NLP tools, including

* Tokenization
* Sentence Split
* The Part-of-Speech (POS) tagger
* Named Entity Recognizer (NER)
* The Coreference Resolution System
* Sentiment Analysis
* Open Information Extraction



# Implementation

## Tokenizer

Tokenizes the text. This component started as a PTB-style tokenizer, but was extended since then to handle both other languages and noisy web-style text. The tokenizer saves the character offsets of each token in the input text, as CharacterOffsetBeginAnnotation and CharacterOffsetEndAnnotation.

## Sentence Split

Splits a sequence of tokens into sentences. (In current CoreNLP, sentence splitting comes after tokenization, and exploits the decisions of the tokenizer.)

## Part Of Speech (POS) Tagging

Label tokens with their POS tag. It Annotate each word in a sentence with a part-of speech. It is Useful for subsequent syntactic parsing. Syntactic analysis uncovers the internal structure of a sentence. Syntactic structure indicates how different words and phrases are related.

Example:

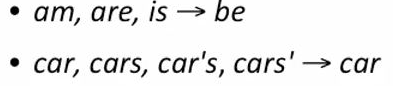
John drove Mary from Austin to Dallas in his Toyota Prius.

[NNP] [VBD] [NNP] [POS] [NNP] [TO] [NNP] [IN] [PRP] [NNP] [NNP]

|  |  |  |
| --- | --- | --- |
| 1 | NN | Noun, singular or mass |
| 2 | IN | Preposition or subordinating conjunction |
| 3 | NNP | Proper noun, singular |
| 4 | POS | Possessive ending |
| 5 | VBD | Verb, past tense |
| 6 | VBP | Verb, non-3rd person singular present |
| 7 | VB | Verb, base form |

## Lemmatization

Generate the word lemmas for all tokens in the corpus. It reduces inflectional forms and sometimes derivationally related forms of a word to a common base form.



Example:

The boy's cars are different colors

The boy car be differ colo

## Named entity recognizer (NER)

Recognizes named entities (person and company names, etc.) in text. The set of entities recognized is language-dependent, and the recognized set of entities is frequently more limited for other languages than what is described below for English. As the name “NERClassifierCombiner” implies, commonly this annotator will run several named entity recognizers and then combine their results. For English, this annotator recognizes named (PERSON, LOCATION, ORGANIZATION, MISC), numerical (MONEY, NUMBER, ORDINAL, PERCENT), and temporal (DATE, TIME, DURATION, SET) entities. Named entities are recognized using a combination of three CRF sequence taggers trained on various corpora, such as CoNLL, ACE and MUC. Numerical entities are recognized using a rule-based system. Numerical entities that require normalization, e.g., dates, have their normalized value stored in NormalizedNamedEntityTagAnnotation.

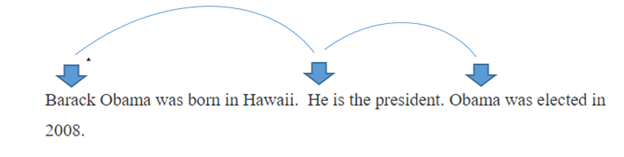
## Coreference resolution system (coref)

The CorefAnnotator finds mentions of the same entity in a text, such as when “Theresa May” and “she” refer to the same person. The annotator implements both pronominal and nominal Coreference resolution. The entire Coreference graph (with head words of mentions as nodes) is saved as a CorefChainAnnotation.

There are three different Coreference systems available in CoreNLP.

1. Deterministic: Fast rule-based Coreference resolution for English and Chinese.
2. Statistical: Machine-learning-based Coreference resolution for English. Unlike the other systems, this one only requires dependency parses, which are faster to produce than constituency parse.
3. Neural: Most accurate but slow neural-network-based Coreference resolution for English and Chinese.

Example:



## Sentiment analysis

Stanford CoreNLP includes the sentiment tool and various programs which support it. The model can be used to analyze text as part of Stanford CoreNLP by adding “sentiment” to the list of annotators. There is also command line support and model training support. Sentiment Annotator implements sentiment model. Attaches a binarized tree of the sentence to the sentence level CoreMap. The nodes of the tree then contain the annotations from RNNCoreAnnotations indicating the predicted class and scores for that subtree.

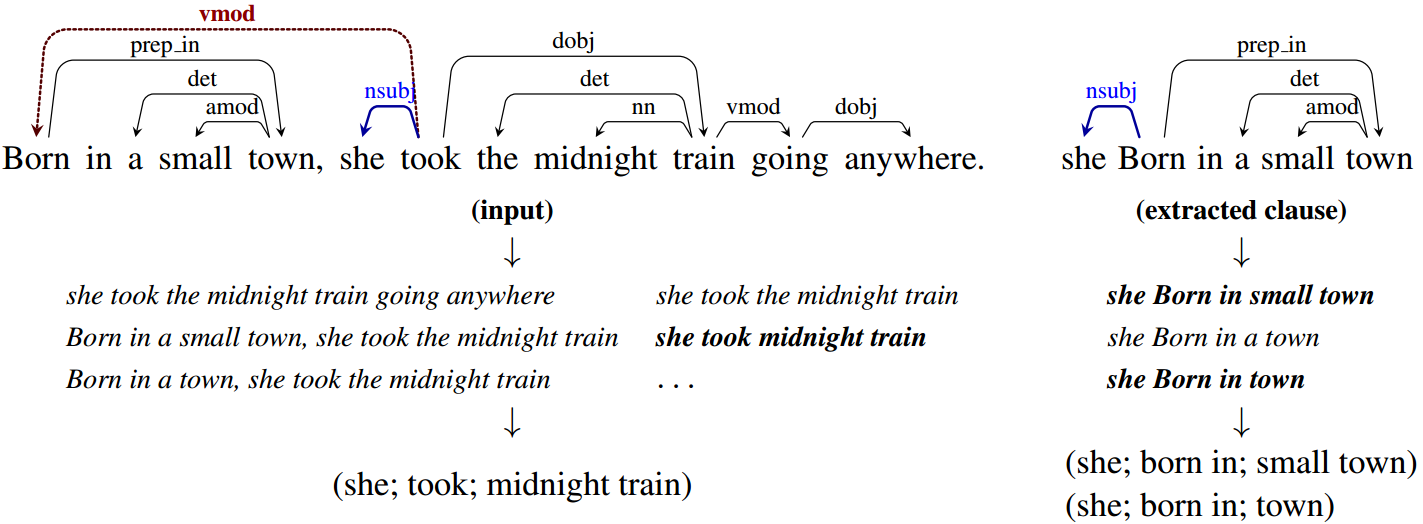
Example: Tom Cruise gave a marvelous performance in MI 2



## Open Information Extraction

Open information extraction (open IE) refers to the extraction of relation tuples, typically binary relations, from plain text. The central difference is that the schema for these relations does not need to be specified in advance; typically the relation name is just the text linking two arguments. For example, Barack Obama was born in Hawaii would create a triple (Barack Obama; was born in; Hawaii), corresponding to the open domain relation was-born-in (Barack-Obama, Hawaii).

The system first splits each sentence into a set of entailed clauses. Each clause is then maximally shortened, producing a set of entailed shorter sentence fragments. These fragments are then segmented into Opine triples, and output by the system. An illustration of the process is given for an example sentence below:



# 

# Applications

* Social media data analysis
* Spam filtering
* Fraud & Risk Management
* Cyber Security
* Knowledge Management
* Customer Service

# Advantages

● Intuitive Syntax

● Lazy Computation

● Fast, Robust

● Maintains Thread Safety

# Limitations & Mitigation

* Some Annotators are slow on processing or take a ton of memory-If you’re using lots of annotators, CoreNLP can easily spend 10–40 seconds just loading an annotation pipeline. So, Don’t run annotators that you don’t need
* Usage of multiple Pipeline - Pipeline loading time can easily dominate actual annotation time. So, if you load a new pipeline frequently, such as for every sentence, then CoreNLP will be painfully slow. You should load an annotation pipeline – what you get when you call new Stanford CoreNLP (props) in code – as infrequently as possible. Often, you can and should just load *one* pipeline and use it for everything. You only need to use multiple pipelines if you simultaneously need different configurations, such as working with multiple human languages or doing processing with different options or annotators.

Beware that some old interfaces to CoreNLP from other programming languages fork a new CoreNLP process every time they are called. Look for a library that either talks to the CoreNLP web service API or directly calls into the Java code and so can avoid creating new annotation pipelines.

# Customized Annotators

Users can add custom annotators to Stanford CoreNLP. In order to create a new annotator we will have to implement Annotator Interface and override its below methods.

**Annotate**

Write annotation logic here

**Requires**

Define prerequisite annotations here

**RequirementsSatisfied**

Validate prerequisite annotations here

Refer to the code example StopwordAnnotation for a demo; this annotator removes stopwords from the given text.

# 

# Running the java source Code

Please refer the attached readme.txt file to execute the source code that were used to demonstrate the functionality different Annotators of as part of presentation

|  |  |  |
| --- | --- | --- |
| Serial No. | Java File(.java) | Implementation of different Annotators |
| 1 | CorefExample.java | Shows the implementation of Annotators like mention and coref |
| 2 | NERExaple.java | Shows the implementations of Annotators like  pos, ssplit, lemma and ner |
| 3 | NaiveBayesClassifier.java | Shows the implementation of NaiveBayes Algorithm using annotators for Preprocessing the corpus.  The accuracy of the model depends on the amount of training data. |
| 4 | MovieReview.java | Show the implementation of Sentiment analysis using Sentiment Annotator.  It classifies the review as positive negative and neutral |
| 5 | StopwordAnnotation | Shows the implementation of Custom Annotators in Stanford CoreNLP |

## 

# Download

Stanford CoreNLP can be downloaded via the link below. This will download a large (~500 MB) zip file containing

(1) The CoreNLP code jar

(2) The CoreNLP models jar (required in your classpath for most tasks),

(3) The libraries required to run CoreNLP, and

(4) Documentation / source code for the project. Unzip this file; open the folder that results and you’re ready to use it.

## Programming languages and operating systems

Stanford CoreNLP is written in Java; recent releases require Java 1.8+.

You can use Stanford CoreNLP from the [command-line](https://stanfordnlp.github.io/CoreNLP/cmdline.html), via its Java [programmatic API](https://stanfordnlp.github.io/CoreNLP/api.html), via [third party APIs](https://stanfordnlp.github.io/CoreNLP/other-languages.html) for most major modern programming languages, or via a [web service](https://stanfordnlp.github.io/CoreNLP/corenlp-server.html). It works on Linux, macOS, and Windows.

## License

Stanford CoreNLP is licensed under the [GNU General Public License](http://www.gnu.org/licenses/gpl.html) (v3 or later; in general Stanford NLP code is GPL v2+, but CoreNLP uses several Apache-licensed libraries, and so the composite is v3+).

More information about the tool can be found [at https://stanfordnlp.github.io/CoreNLP/](at%20https:/stanfordnlp.github.io/CoreNLP/)

**References:**

<https://stanfordnlp.github.io/CoreNLP/>